

AMENDMENTS TO THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

- 1 1. (Currently amended) A method for using a hash table that is fully
2 dynamic and lock-free, comprising:
3 performing a lookup into the hash table, wherein the lookup involves,
4 using a hash key to lookup a bucket pointer in a bucket
5 array,
6 following the bucket pointer to a data node within a linked
7 list containing all of the data nodes in the hash table, and
8 searching from the data node through the linked list to
9 locate a node that matches the hash key if one exists;
10 wherein the linked list contains only data nodes and at most a constant
11 number of dummy nodes; and
12 if the average number of data nodes in each bucket exceeds a maximum
13 value:
14 increasing the number of buckets in the bucket array to
15 form a larger bucket array, and
16 using more bits from the hash key to perform lookups in the
17 larger bucket array,
18 wherein the data nodes are stored in the linked list in bit-
19 inverted hash key order, and

20 wherein increasing the number of buckets in the bucket
21 array involves mapping the existing bucket array into the top half
22 of the larger bucket array.

1 2. (Original) The method of claim 1, wherein the data node pointed to by
2 the bucket pointer precedes the nodes in the bucket.

1 3. (Original) The method of claim 1, wherein deleting the data node from
2 the linked list involves:
3 using an atomic operation to mark the data node as dead; and
4 atomically updating the next pointer of the predecessor of the data node to
5 point around the data node to the successor of the data node in the linked list.

1 4. (Currently amended) The method of claim 3~~-claim 2~~, wherein deleting
2 the data node from the linked list additionally involves redirecting the next pointer
3 of the data node to become a back pointer that points to the predecessor of the
4 data node.

1 5. (Original) The method of claim 4, wherein if a search through a chain of
2 nodes from the back pointer does not lead to a live node, the method further
3 comprises:
4 obtaining a parent bucket pointer;
5 searching through the linked list from a node pointed to by the parent
6 bucket pointer to locate a starting node for the bucket pointer; and
7 updating the bucket pointer to point to the starting node.

1 6. (Original) The method of claim 2, wherein deleting the data node from
2 the linked list involves using garbage collection or a solution to the repeat
3 offender problem to reclaim the data node if possible.

1 7. (Original) The method of claim 1, further comprising generating the
2 hash key by performing a pre-hashing operation to achieve a uniform distribution
3 of hash keys over possible hash key values.

1 8 (Canceled).

1 | 9. (Currently amended) The method of claim 1-~~claim 8~~, wherein buckets in
2 the larger bucket array are initialized on-the-fly as they are referenced.

1 | 10. (Currently amended) The method of claim 1-~~claim 8~~, wherein
2 initializing a bucket pointer involves:
3 obtaining a parent bucket pointer for the bucket pointer;
4 searching through the linked list from a node pointed to by the parent
5 bucket pointer to locate a starting node for the bucket pointer; and
6 updating the bucket pointer to point to the starting node.

1 11. (Original) The method of claim 1, wherein if there exists an old hash
2 table, initializing a bucket pointer involves looking for a corresponding entry in
3 the old hash table first, and if this fails:
4 obtaining a parent bucket pointer for the bucket pointer;
5 searching through the linked list from a node pointed to by the parent
6 bucket pointer to locate a starting node for the bucket pointer; and
7 updating the bucket pointer to point to the starting node.

1 12 (Canceled).

1 13. (Currently amended) The method of claim 1 ~~claim 8~~,
2 wherein the data nodes are stored in the linked list in hash key order; and
3 wherein increasing the number of buckets in the bucket array involves
4 interleaving the bucket array into the larger bucket array.

1 14. (Original) The method of claim 2, wherein if the average number of
2 data nodes in each bucket falls below a minimum value, the method further
3 comprises:
4 reducing the number of buckets in the bucket array to form a smaller
5 bucket array; and
6 using fewer bits from the hash key to perform lookups in the smaller
7 bucket array.

1 15. (Currently amended) A computer-readable storage medium storing
2 instructions that when executed by a computer cause the computer to perform a
3 method for using a hash table that is fully dynamic and lock-free, the method
4 comprising:
5 performing a lookup into the hash table, wherein the lookup involves,
6 using a hash key to lookup a bucket pointer in a bucket
7 array,
8 following the bucket pointer to a data node within a linked
9 list containing all of the data nodes in the hash table, and
10 searching from the data node through the linked list to
11 locate a node that matches the hash key if one exists;
12 wherein the linked list contains only data nodes and at most a constant
13 number of dummy nodes; and

14 if the average number of data nodes in each bucket exceeds a maximum
15 value:
16 increasing the number of buckets in the bucket array to
17 form a larger bucket array, and
18 using more bits from the hash key to perform lookups in the
19 larger bucket array,
20 wherein the data nodes are stored in the linked list in bit-
21 inverted hash key order, and
22 wherein increasing the number of buckets in the bucket
23 array involves mapping the existing bucket array into the top half
24 of the larger bucket array.

1 16. (Original) The computer-readable storage medium of claim 15,
2 wherein the data node pointed to by the bucket pointer precedes the nodes in the
3 bucket.

1 17. (Original) The computer-readable storage medium of claim 15,
2 wherein deleting the data node from the linked list involves:
3 using an atomic operation to mark the data node as dead; and
4 atomically updating the next pointer of the predecessor of the data node to
5 point around the data node to the successor of the data node in the linked list.

1 18. (Currently amended) The computer-readable storage medium of claim
2 17-claim 16, wherein deleting the data node from the linked list additionally
3 involves redirecting the next pointer of the data node to become a back pointer
4 that points to the predecessor of the data node.

1 19. (Original) The computer-readable storage medium of claim 18,
2 wherein if a search through a chain of nodes from the back pointer does not lead
3 to a live node, the method further comprises:
4 obtaining a parent bucket pointer;
5 searching through the linked list from a node pointed to by the parent
6 bucket pointer to locate a starting node for the bucket pointer; and
7 updating the bucket pointer to point to the starting node.

1 20. (Original) The computer-readable storage medium of claim 16,
2 wherein deleting the data node from the linked list involves using garbage
3 collection or a solution to the repeat offender problem to reclaim the data node if
4 possible.

1 21. (Original) The computer-readable storage medium of claim 15,
2 wherein the method further comprises generating the hash key by performing a
3 pre-hashing operation to achieve a uniform distribution of hash keys over possible
4 hash key values.

1 22 (Canceled).

1 | 23. (Currently amended) The computer-readable storage medium of claim
2 | 15-claim-22, wherein buckets in the larger bucket array are initialized on-the-fly as
3 | they are referenced.

1 | 24. (Currently amended) The computer-readable storage medium of claim
2 | 15-claim-22, wherein initializing a bucket pointer involves:
3 | obtaining a parent bucket pointer for the bucket pointer;

4 searching through the linked list from a node pointed to by the parent
5 bucket pointer to locate a starting node for the bucket pointer; and
6 updating the bucket pointer to point to the starting node.

1 25. (Original) The computer-readable storage medium of claim 15,
2 wherein if there exists an old hash table, initializing a bucket pointer involves
3 looking for a corresponding entry in the old hash table first, and if this fails:
4 obtaining a parent bucket pointer for the bucket pointer;
5 searching through the linked list from a node pointed to by the parent
6 bucket pointer to locate a starting node for the bucket pointer; and
7 updating the bucket pointer to point to the starting node.

1 26 (Canceled).

1 27. (Currently amended) The computer-readable storage medium of claim
2 15-claim 22,
3 wherein the data nodes are stored in the linked list in hash key order; and
4 wherein increasing the number of buckets in the bucket array involves
5 interleaving the bucket array into the larger bucket array.

1 28. (Original) The computer-readable storage medium of claim 15,
2 wherein if the average number of data nodes in each bucket falls below a
3 minimum value, the method further comprises:
4 reducing the number of buckets in the bucket array to form a smaller
5 bucket array; and
6 using less bits from the hash key to perform lookups in the smaller bucket
7 array.

1 29. (Currently amended) An apparatus that implements a hash table that is
2 fully dynamic and lock-free, comprising:
3 a lookup mechanism, wherein the lookup mechanism is configured to,
4 use a hash key to lookup a bucket pointer in a bucket array,
5 follow the bucket pointer to a data node within a linked list
6 containing all of the data nodes in the hash table, and to
7 search from the data node through the linked list to locate a
8 node that matches the hash key if one exists;
9 wherein the linked list contains only data nodes and at most a constant
10 number of dummy nodes; and
11 a bucket array expansion mechanism, wherein if the average number of
12 data nodes in each bucket exceeds a maximum value, the bucket array expansion
13 mechanism is configured to:
14 increase the number of buckets in the bucket array to form a
15 larger bucket array, and to
16 use additional bits from the hash key to perform lookups in
17 the larger bucket array,
18 wherein the data nodes are stored in the linked list in bit-
19 inverted hash key order, and
20 wherein the bucket array expansion mechanism is
21 configured to map the existing bucket array into the top half of the
22 larger bucket array.

1 30. (Original) The apparatus of claim 29, wherein the data node pointed to
2 by the bucket pointer precedes the nodes in the bucket.

1 31. (Original) The apparatus of claim 29, further comprising a node
2 deletion mechanism configured to:

3 use an atomic operation to mark the data node as dead; and to
4 atomically update the next pointer of the predecessor of the data node to
5 point around the data node to the successor of the data node in the linked list.

1 | 32. (Currently amended) The apparatus of claim 31-~~claim 30~~, wherein the
2 node deletion mechanism is additionally configured to redirect the next pointer of
3 the data node to become a back pointer that points to the predecessor of the data
4 node.

1 33. (Original) The apparatus of claim 32, further comprising a bucket
2 pointer updating mechanism, wherein if a search through a chain of nodes from
3 the back pointer does not lead to a live node, the bucket pointer updating
4 mechanism is configured to:
5 obtain a parent bucket pointer;
6 search through the linked list from a node pointed to by the parent bucket
7 pointer to locate a starting node for the bucket pointer; and to
8 update the bucket pointer to point to the starting node.

1 34. (Original) The apparatus of claim 30, wherein the node deletion
2 mechanism is additionally configured to use garbage collection or a solution to the
3 repeat offender problem to reclaim the data node if possible.

1 35. (Original) The apparatus of claim 29, further comprising a pre-hashing
2 mechanism configured to generate the hash key by performing a pre-hashing
3 operation to achieve a uniform distribution of hash keys over possible hash key
4 values.

1 36 (Canceled).

1 | 37. (Currently amended) The apparatus of claim 29~~-claim 36~~, further
2 | comprising a bucket initialization mechanism configured to initialize buckets in
3 | the larger bucket array on-the-fly as they are referenced.

1 | 38. (Currently amended) The apparatus of claim 29~~-claim 36~~, wherein the
2 | bucket initialization mechanism is configured to:
3 | obtain a parent bucket pointer for the bucket pointer;
4 | search through the linked list from a node pointed to by the parent bucket
5 | pointer to locate a starting node for the bucket pointer; and to
6 | update the bucket pointer to point to the starting node.

1 | 39. (Original) The apparatus of claim 29, wherein if there exists an old
2 | hash table, the bucket initialization mechanism is configured to look for a
3 | corresponding entry in the old hash table first, and if this fails to:
4 | obtain a parent bucket pointer for the bucket pointer;
5 | search through the linked list from a node pointed to by the parent bucket
6 | pointer to locate a starting node for the bucket pointer; and to
7 | update the bucket pointer to point to the starting node.

1 | 40 (Canceled).

1 | 41. (Currently amended) The apparatus of claim 29~~-claim 36~~,
2 | wherein the data nodes are stored in the linked list in hash key order; and
3 | wherein the bucket array expansion mechanism is configured to interleave
4 | the bucket array into the larger bucket array.

1 | 42. (Original) The apparatus of claim 30, further comprising a bucket array
2 | contraction mechanism, wherein if the average number of data nodes in each

3 bucket falls below a minimum value, the bucket array contraction mechanism is
4 configured to:
5 reduce the number of buckets in the bucket array to form a smaller bucket
6 array; and to
7 use less bits from the hash key to perform lookups in the smaller bucket
8 array.